SWTPC CORES ASS MEM; START = 0020
MEM; END = 23FF
START OP 0100 COLD
0103 WARM

8K BASIC VERSION 2.0©

USERS GUIDE

WRITTEN BY

ROBERT H. UITERWYK

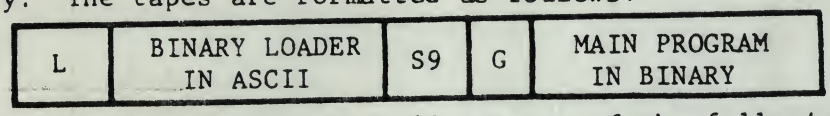
4402 Meadowwood Way

Tampa, Florida 33624

SOUTHWEST TECHNICAL PRODUCTS CORPORATION
219 West Rhapsody San Antonio, Texas 78216

NOTICE TO USERS OF SWTPC PAPER AND CASSETTE TAPES

In order to help reduce the time necessary to load programs through either a paper tape reader or an SWTPC AC-30 cassette interface, the longer tapes supplied from SWTPC will be furnished in a binary format instead of the conventional ASCII. At the beginning of each tape is a binary loader program that will load into the computer using the regular ASCII format. The program then executes itself and loads the main program in binary. Using this method tapes will load in approximately 1/3 normal time. When using an SWTPC AC-30 lock the reader in the ON position and type L. For paper tape readers either lock the reader on or re-set the reader-on relay if the load stops. (the computer will send a reader off character after an ASCII S9 on the tape is loaded in) On cassette tapes, one side will be in conventional ASCII (side with long leader) and one side will be in binary. The tapes are formatted as follows:

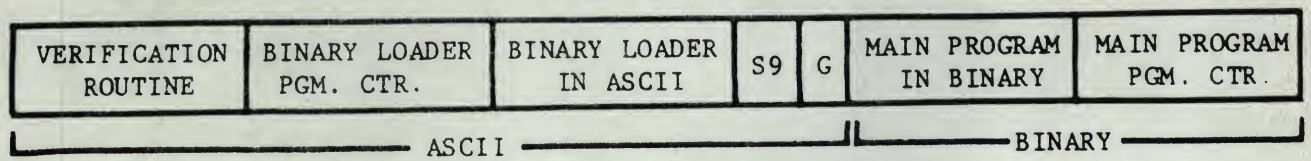


As the tape loads, you will see one of the following displays on your terminal: (either is OK)

```
*L                    *L
*G                    *??
* (register dump)     * (register dump)
```

Some tapes may have an additional feature which will verify that the tape loaded correctly into memory. If, after loading the tape, you find that the program counter is not automatically set to the correct value then you probably have a verifying tape. If this is the case simply typing a G will automatically check the validity of the program and execute it. If the message LOAD ERROR is displayed then the tape did not load correctly into memory. The most common cause of this is a memory problem - there can be problems that MEMCON and ROBIT will not find.

The format for a self verifying tape is as follows:



As before, one side of a cassette tape will be in binary and the other side in ASCII.

If you are unable to load a tape please check the following:

1) Be sure the reader is locked on to load a binary tape.
2) Try different volume and tone control settings.
3) Clean your tape heads with alcohol and a cotton swab.
4) Re-check all memory if a LOAD ERROR is displayed.

FEATURES OF SWTPC 8K BASIC © VERSION 2.0

* All mathematical operations are performed in BCD (Binary Coded Decimal) arithmetic for maximum accuracy.

* User programs may be saved and loaded from either SWTPC AC-30 Cassette or paper tape and may be filed to allow several short programs on one tape.

* Most function subprograms including transcendentals are implemented.

* String variables and arrays are allowed.

* Most program statements may be executed in the direct mode (no statement numbers for immediate calculations and enhanced program debugging).

* Most programs will run with a memory of only 12K bytes.

* Users can call machine language programs with the USER Function.

* Multiple statements per line are allowed.

Notation

In this manual square brackets ([ ]) are used to denote options.

statement n	means	statement number
var	means	variable name
exp	means	mathematical expression
rel exp	means	relational expression
"textstring"	means	a collection of literal alpha-numeric characters enclosed by quotation marks

Program Structure

A BASIC program is comprised of statements. Every statement begins with a statement number, followed by the statement body, and terminated by a carriage return.

There are four types of statements in BASIC:

Declarations, Assignments, Input/Output and Control.

These statement types are described in the corresponding sections of this manual.

Statements

- Every statement must have a statement number ranging between 1 and 9999. Do not use line number Ø.

- Statement numbers are used by BASIC to order the program statements sequentially.

- In any program, a statement number can be used only once.

- A previously entered line may be changed by entering the same line number along with the desired statement. Typing a line number followed immediately by a Carriage Return deletes that line number.

- Statements need not be entered in numerical order, because BASIC will automatically order them in ascending order.

- A statement may contain no more than 72 characters including blanks.

- Blanks, unless within a character string and enclosed by quotation marks, are <u>not</u> processed by BASIC, and their use is optional.

Example: 110 LET A=B + (3.5*5E2)
is exactly equivalent to:
110LETA=B+(3.5*5E2)

- With blanks, the statement is more readable, but takes longer to process; however, numbers can contain no imbedded blanks.

Multiple statement lines are accepted. A colon (:) should be used as the seperator. BASIC will process the line from left to right. Example:

```
10 A=3 : B=5 : C=A*B . PRINT C/A    is equivalent to:

10 A=3
20 B=5
30 C=A*B
40 PRINT C/A
```

Data Format

The range of numbers that can be represented in this version of BASIC is:

1.0E-99 to 9.99999999E+99

E+99 represents 10^{99} while E-99 represents 10^{-99}. The E stands for exponent.

There are nine digits of significance in this version of BASIC. Numbers are internally truncated (last digits dropped) to fit this precision.

Numbers may be entered and displayed in three formats: integer, decimal and exponential.

Example: 153 34.52 136E-2

Variable Names

Variables may be named any single alphabetic character or any single alphabetic character followed by a number 0 thru 9. String variables may be any single variable followed by $.

Example: A, B, C, A5, X6, A$, Z$

String Variable

Any single letter, or subscripted letter followed by a "$". (A4$ is not valid). A string variable may contain a maximum of 32 characters.

Note: A string variable reserves 32 bytes even if only one character is stored within it.

Example: A$, C$, F$, A$(3), F$(1), G$(9,9)

Note: These are distinct from numeric variables of the same name ...For instance, A=3, A$="HELLO", A(3)=7, and A$(3)="GOODBYE" are all valid.

REM

The REM, or remark statement, is a non-executable statement which has been provided for the purpose of making program listings more readable. By generous use of REM statements, a complex program may be more easily understood. REM statements are merely reproduced on the program listing, they are not executed. If control is given to a REM statement, it will perform no operation. (It does however, take a finite amount of time to process the REM statement.)

Program Preparation

After BASIC is loaded into your system, it may be started at memory address $\emptyset 1\emptyset\emptyset_{16}$. At this time, BASIC will automatically determine the range

of working storage. If you wish to limit the amount of memory BASIC uses, refer to Appendix F of this manual.

The system is then ready to accept commands or statements. For example the user might enter the following program:

```
150 REM DEMOSTRATION
160 PRINT "ENTER A NUMBER";
170 INPUT A
180 LET P = A*A*3.1415926
185 PRINT
190 PRINT "THE AREA OF A CIRCLE WITH";
200 PRINT "RADIUS"; A; "IS"; P
210 STOP
```

If the user wishes to insert a statement between two others, he need only type a statement number that falls between the other two. For example:

```
183 REM THIS IS INSERTED BETWEEN 180 and 185.
```

If it is desired to replace a statement, a new statement is typed that has the same number as the one to be replaced. For example:

```
180 P=(A*A)*3.1415926 replaces previous LET statement.
```

Each line entered is terminated by a Carriage Return. BASIC positions the print unit to the correct position on the next line.

The control 0 and control X control characters may be used to backspace one character or delete a line that was typed in error. See explanation in the Commands Section.

If the user wishes to execute the program at this point, the RUN command should be entered.

Commands

It is possible to communicate in BASIC by typing direct commands at the terminal device. Also, certain other statements can be directly executed when they are entered without statement numbers.

Commands have the effect of causing BASIC to take immediate action. A typical BASIC language program, by contrast, is first entered statement by statement into the memory and then executed only when the RUN command is given.

When BASIC is ready to receive commands, the word READY is displayed

on the terminal device. After each entry, the system will prompt with a "#".

Commands are typed without statement numbers. After a command has been executed READY will again be displayed indicating that BASIC is ready for more input, either another command or program statements.

Note: The LIST, SAVE, LOAD and APPEND commands all have #N appended to them, where "N" is the port number where the I/O operation will take place (See Input/Output Commands).

LIST [statement m], [statement n]

Causes the statements of the current program to be displayed on the user's terminal. The lines are listed in increasing numerical order by statement number. The display will be only statement number m, if given, or statements m through n, if given, or all statements if no argument is given.

Examples:
```
LIST 30
LIST 20, 200
LIST #7, 30, 70
LIST #0
```

RUN

Causes the current program resident in memory to begin execution at the first statement number. Run always begins at the lowest statement number. Run resets all program parameters and initializes all variables to zero.

CONT

Entered after a "STOP" has been encountered, or after a Control C (Break) has been entered. Only to be used if an error was not encountered, and if the program has not been changed.

NEW

The NEW (scratch or clear) command causes working storage and all variables and pointers to be reset. The effect of this command is to erase all traces of the program from memory and to start over. This command also sets LINE to 48 and DIGITS to 0.

SAVE

Causes the program in memory to be saved on either a SWTPC AC-30 cassette interface or a paper tape punch. Control commands are output to control the read/record mechanism. Complete details are given in Appendix C.

on the terminal device. After each entry, the system will prompt with a
"?".

Commands are typed without statement numbers. After a command has
been executed EDASM will again be displayed indicating that BASIC is ready
for more input, either another command or program statements.

Note: The LIST, SAVE, LOAD and APPEND commands will have ON accepted
       in them, where "ON" is the port number where the I/O operation will
       take place (See Input/Output Commands).

LIST [statement #], [statement #]

Causes the enumeration of the current program to be displayed on the
users terminal. The lines are listed in increasing numerical order by
statement number. The display will be only statement number 0, if given,
or statements 0 through n, if given, or all statements if no argument is
given.

Examples: LIST 30
           LIST 30, 200
           LIST 2, 30, 70
           LIST 40

EDIT

Causes the control program resident in memory to begin execution at
the first statement number. One always begins at the lowest statement
number. One resets all program parameters and initializes all variables
to zero.

CONT

Entered after a "STOP" has been encountered, or after a Control C
(break) has been entered, only to be used if no error has occurred,
and if the program has not been changed.

NEW

The NEW Command or clear command causes wiping out all
variables and pointers to be reset. The effect of this command is to erase
all traces of the program from memory and to start over. This command
also sets LINE to 32 and DIGITS to 0.

SAVE

Causes the program in memory to be saved on tape a SAVEC #2-30
cassette interface or a paper tape punch. Control characters are output
to control the reinforced mechanism. Complete details are given in
Appendix C.

LOAD

Causes a tape (magnetic or paper) that was previously "Saved" to be loaded into memory. It clears out the present memory (if any) before starting. Complete details are given in Appendix C.

APPEND

Works exactly like LOAD, but does not clear out present contents of memory.

CONTROL C

Pressing the Control C key on the terminal console will cause BASIC to halt its current operation and to respond with a READY. BASIC will then accept further commands. This command is often used to stop a LIST command before it has completed or to halt the execution of a looped program. Due to the use of a PIA on the control interface, the user may have to type Control C several times.

CONTROL X

Clear the current line buffer. If the user types a line at the terminal and decides that the line is in error and should be deleted, a simultaneous depression of the Control and X keys before the carriage return will clear the line. The system responds with "DELETED" and a CR and LF.

CONTROL O

Single character backspace. If a character is determined to have been typed in error, it may be deleted by simultaneously pressing the Control and O keys, then entering the correct character. A _ is echoed to signify the backspace. You may backspace as many character positions as necessary. BASIC will prevent you from backing past the start of the line.

PATCH

Causes the computer to return to the Mikbug® operating system and outputs a Carriage Return, Line Feed and '*' on the print device. If no BASIC memory and the program counter addresses (AØ48 and AØ49) are not changed, typing a G for "Go to User Program" will restart the program with the User program intact. The PATCH command may even be inserted as a control statement within a BASIC user program. When the PATCH statement is encountered, control is transferred to Mikbug® and the computer outputs a Carriage Return, Line Feed and '*'. Typing a G retruns control back to the BASIC user program statement immediately following the PATCH statement.

TRACE ON

Prints each Line number as the line is executed. A debugging tool!

LOAD

Causes a tape (magnetic or paper) that was previously "Saved" to be loaded into memory. It clears out the present memory (if any) before starting. Complete details are given in Appendix C.

APPEND

Works exactly like LOAD, but does not clear out present contents of memory.

CONTROL C

Pressing the Control C key on the terminal console will cause BASIC to halt the current operation and to return to CMAIN. BASIC will then accept further commands. This command is often used to stop a LIST command before it has finished or to halt the execution of a looped program. Not all of the CTRL or ESC on the control terminal, the user may have to type Control C several times.

CONTROL X

Clears the current line buffer. If the user types a line at the terminal and decides that he does not want what he has typed to be defined, a simultaneous depression of the Control and X keys before the carriage return will clear the line. The system responds with "@\#!???" and a CR and LF.

CONTROL O

Single character backspace. If a character is determined to have been typed in error, it may be deleted by simultaneously pressing the Control and O keys, then entering the correct character. A _ is echoed to signify the backspace. You may backspace as many characters as necessary. Note it will prevent you from backing past the start of the line.

PATCH

Causes the computer to return to the MITRE® operating system and outputs a Control Return, Line Feed and '#' on the CRT device. If no BASIC memory and one or more character addresses (APAD and /APAD) are not changed, typing a C for "Do no User Program" will restart the program with the User program intact. The PATCH command may also be inserted in a User stored program. When the PATCH statement is encountered, control is transferred to MITRE® and the computer outputs a Control Return, Line Feed and '#'. Typing a C returns control back to the BASIC user program statement immediately following the PATCH statement.

TRACE ON

Prints each line number as the line is executed. A debugging tool.

TRACE OFF

Stops the "TRACE" function.

LINE

Specifies the number of print positions in a line (Line Length).

Example: LINE=65, LINE=80, LINE=40

Note: Each line is broken up into 16 character "Zones". If the print position is within the last 25 percent of the "line" length and a "space" is printed, a C/R LF will be output. This is so that a number or word will not be split up at the end of a print line. If you wish to inhibit this (for precise print control) set LINE equal to 0.

DIGITS

Sets the number of digits printed to the right of the decimal point. This will truncate (not round) any digits greater than the number printed, and will force "0"s if there aren't enough significant digits to fill the number of positions specified in the "DIGITS" command.

DIGITS=0 resets to floating point mode.

PORT

PORT=X defines the computer I/O Port which will serve as the 'Control Port'. "X" can be a constant, variable, or expression.

Warning - If you define a Port without a terminal as the Control Port, all messages (including the "Ready") will be inputed and outputed from that Port...therefore you will lose control of your program!

NOTE: PIA ports require handshaking, If handshaking is not available, then you must use the PEEK command to examine the PIA registers. Also, BASIC will always accept a break from port 1, therefore never leave port 1 without a terminal connected.

PORT	TYPE OF PORT
0	ACIA
1	MODIFIED PIA (CONTROL PORT)
2	ACIA
3	ACIA
4	PIA
5	PIA
6	PIA
7	PIA (LINE PRINTER, BY CONVENTION, SWTPC PR-40)

POKE

POKE (I,J) takes the decimal value of "J" and places it in the decimal memory location "I". WARNING - Exercise great care in the use of this

command, as it is easy to bomb the basic interpreter, your program, and/ or your data!

DIRECT EXECUTION - CALCULATOR MODE

BASIC facilitates computer utilization for the immediate solution of problems, generally of a mathematical nature, which do not require iterative program procedures. To clarify: BASIC may be used as a sophisticated electronic calculator by means of its "direct" (no statement number) statement execution capability.

While BASIC is in the command mode some BASIC statements may be entered without statement numbers. BASIC will immediately execute such statements. This is called the direct mode of execution. Example:

```
PRINT    (28 + 3.75)  *  2.317
```

Statements that are entered with statement numbers are considered to be program statements which will be executed later.

In the following sections of this manual all BASIC statements are described. Only those statements which are flagged with the word 'Direct' may be used in the direct mode.

Another use for direct execution is as an aid in program development and debugging. Through use of direct statements, program variables can be altered or read, and program flow may be directly controlled.

DIM var (exp) or var (exp),var(exp) or var(exp,exp)

The DIM statement allocates memory space for an array. In this version of BASIC, one or two dimension arrays are allowed. Maximum array size is 255 x 255 elements. All array elements are set to zero by the DIM statement.

If an array is not explicitly defined by a DIM statement, it is assumed to be defined as an array of 10 elements (or 10 X 10 if two elements are used) upon first reference to it in a program.

Caution: An array can be determined only once in a program, implicitly and explicitly. Also only the variables A thru Z (followed by $)may be dimensioned for strings.

```
Example: DIM  A(10), C(R5+8), D(30,A*3), A7(20), C$(30), Z$(5)
              but not A6$(5)
```

DATA num [num,..num]

READ var [var,..,var]

RESTORE

The DATA and READ statements are used in conjunction with each other

as one of the methods to assign values to variables. Every time a DATA statement is encountered, the values in the argument field are assigned sequentially to the next available position of a data buffer. All DATA statements, no matter where they occur in a program, cause DATA to be combined into one list.

READ statements cause values in the data buffer to be accessed sequentially and assigned to the variables named in the READ statement. They start with the first data element from the first data statement, then the second element, to the end of the first data statement, then to the first element of the second data statement, etc., each time a READ command is encountered. If a READ is executed, and the DATA statements are out of data, an error is generated.

Numeric and string data may be intermixed, however it must be called in the appropriate order.

Note: String data need not be enclosed within quotes (") as the comma (,) acts as the delimiter. However, if the string contains a (,), then it must be delimited by quotes ("). Example:

```
10 DATA 10,20,30,56.7,"TEST, ONE",1.67E30,8,HELLO
20 READ A,B,C,D,E$,F,G5,F$
```

Note: DATA STATEMENTS may be placed anywhere within the program.

Example:
```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B,C,D,E
```

Is the equivalent of:

```
10 LET B=1
20 LET C=2
30 LET D=3.5
40 LET E=4.5
```

The RESTORE statement causes the data buffer pointer, which is advanced by the execution of READ statements, to be reset to point to the first position in the data buffer.

Example:
```
110 DATA 1,2,3.5
120 DATA 4.5,7,70
130 DATA 80,81
140 READ B,C
150 RESTORE
160 READ D,E
```

In this example, the variables would be assigned values equal to:

```
100 LET B=1
101 LET C=2
102 LET D=1
103 LET E=2
```

Assignment Statements

LET var=exp (Direct)

The LET statement is used to assign a value to a variable. The use of the word LET is optional unless you are in the direct mode.

```
Example:  100 LET B=827
          110 LET C=87E2
          120 R=(X*Y)/2*A
          130 C$="THIS IS TEXT"
```

The equal sign does not mean equivalence as in ordinary mathematics. It is the replacement operator. It says: replace the value of the variable named on the left with the value of the expression on the right. The expression on the right can be a simple numerical value or an expression composed of numerical values, variables, mathematical operators, and functions.

MATHEMATICAL OPERATORS

The mathematical operators used to form expressions are:

```
↑ ..... Exponentiation - Raises to a power
- (unary) ..... Negate (Requires only one operand)
* ..... Multiplication
/ ..... Division
+ ..... Addition
- ..... Subtraction
```

No two mathematical operators may appear in sequence, and no operator is ever assumed: A++B and (A+2) (B-3) are not valid. Exception: A↑-3 is allowed.

PRIORITY OF OPERATIONS

1.	Exponentiation	(↑)			
2.	Negation	(-)			
3.	Multiplication	(*)	and	Division	(/)
4.	Addition	(+)	and	Subtraction	(-)

The expression is evaluated left to right in the above priority sequence unless parenthesis are encountered (the operators within the parenthesis are evaluated first, utilizing the above priority structure.). Example:

```
A=2
B=3
C=4

B=C/2=5
```

```
B↑2+C/A↑2=10
C↑2-C/A=14
A*( A+B*2 )-22=0
A↑A↑B=64
```

STRING CONCATENATION

Although any one string variable may be a maximum of 32 characters long, strings may be concatenated (joined) up to a maximum of 128 characters (for printing). The concatenation symbol is the "+".

```
Example:  A$="HELLO"
          B$="JOHN"
          C$=A$+B$
          C$="HELLOJOHN"
```

Control Statements

Control statements are used to control the natural sequential progression of program statement execution. They can be used to transfer control to another part of a program, terminate execution, or control iterative processes (loops).

FOR var=exp1 TO exp2 STEP exp3
.
.
.
NEXT var

The FOR and NEXT statements are used together for setting up program loops. A loop causes the execution of one or more statements for a specified number of times. The variable in the FOR_TO statement is initially set to the value of the first expression (exp1). Subsequently, the statements following the FOR are executed. When the NEXT statement is encountered, the named variable is added to the value indicated by the STEP expression in the FOR_TO statement, and execution is resumed at the statement following the FOR_TO. If the addition of the STEP value develops a sum that is greater than the TO expression (exp2) or, if STEP is negative, a sum less than the TO expression (exp2) , the next instruction executed will be the one following the NEXT statement. If no STEP is specified, a value of one is assumed. If the TO value is initially less than the initial value, the FOR_NEXT loop will still be executed once.

```
Example:  110 FOR I=.5 TO 10
          120 INPUT X
          130 PRINT I,X,X/5.6
          140 NEXT I
```

Although expressions are permitted for the initial, final, and STEP

values in the FOR statement, they will be evaluated only once, the first time the loop is entered.

It is not possible to use the same indexed variable in two loops if they are nested.

When the statement after the NEXT statement is executed, the variable is equal to the value that caused the loop to terminate, not the TO value itself. In the example, I would be equal to 9.5 when the loop terminates.

STOP

The STOP statement causes the program to stop executing. BASIC returns to the command mode. The STOP statement differs from the END statement in that it causes BASIC to display the statement number where the program halted, and the program can be restarted by a GOTO or a CONT. The message displayed is STOP XXXX.

END

The END statement causes the program to stop executing. BASIC returns to the command mode. In this version of BASIC, END may appear more than once and need not appear at all.

GOTO statement n (Direct)

The GOTO statement directs BASIC to execute the specified statement unconditionally. Program flow continues from the new statement.

Example: 150 GOTO 270

GOSUB statement n

A subroutine is a sequence of instructions which perform some task that would have utility in more than one place in a BASIC program. To use such a sequence from more than one place, BASIC provides subroutines and functions.

A subroutine is a program unit that receives control upon execution of a GOSUB statement. Upon completion of the subroutine, control is returned to the statement following the GOSUB by execution of a RETURN statement.

A function is a program unit to which control is passed by a reference to the function name in an expression. A value is computed for the function name, and control is returned to the statement that invoked the function.

```
GOSUB statement n
.
.
.
statement n
.
.
.
```

values to the FOR statement, they will be evaluated only once, the first time the loop is entered.

It is not possible to use the same indexed variable in two loops if they are nested.

When the statement after the NEXT statement is executed, the variable is equal to the value that caused the loop to terminate, not the TO value itself. In the example, I would be equal to 21 when the loop terminates.

STOP

The STOP statement causes the program to stop executing. BASIC returns to the command mode. The STOP statement differs from the END statement in that it causes BASIC to display the statement number where the program halted, and the program can be restarted by a CONT or a GOTO. The message displayed is STOP AT###.

END

The END statement causes the program to stop executing. BASIC returns to the command mode. In this version of BASIC, END may appear more than once and need not appear at all.

GOTO statement n (Direct)

The GOTO statement directs BASIC to execute the specified statement unconditionally. Program flow continues from the new statement.

Example: 150 GOTO 210

GOSUB statement n

A subroutine is a sequence of instructions which performs some task that would have utility in more than one place in a BASIC program. It may be both a sequence from more than one place, BASIC performs subroutines and functions.

A subroutine is a unique unit that receives control upon execution of a GOSUB statement. Upon completion of the subroutine, control is returned to the statement following the GOSUB by execution of a RETURN statement.

A function is a program unit to which control is passed by a reference to the function name in an expression. A value is returned for the function name, and control is returned to the statement that invoked the function.

GOSUB statement n

.
.
.
statement n
.
.
.

RETURN

The GOSUB statement causes control to be passed to the given statement number. It is assumed that the given statement number is an entry point of a subroutine. The subroutine returns control to the statement following the GOSUB statement with a RETURN statement.

Subroutine
Example:

```
100 X=1
110 GOSUB 200
120 PRINT X
125 X=5.1
130 GOSUB 200
140 PRINT X
150 STOP
200 X=(X+3)*5.32E3
210 RETURN
211 END
```

Subroutines may be nested; that is one subroutine may use GOSUB to call another subroutine which in turn can call another. Subroutine nesting is limited to eight levels.

ON EXP GOTO statement n, (m,...L)

ON EXP GOSUB statement n, (m,...L)

This statement transfers control to the statement or subroutine as defined by the value of exp. The expression will be evaluated, truncated (chopped off after the decimal point) and control then transferred to the nth statement number (where n is the integer value of the expression).

Example: ON N GOTO 110,300,500,900

Means:
```
If N<1 You will get an error
If N=1 GOTO 110
If N=2 GOTO 300
If N=3 GOTO 500
If N=4 GOTO 900
If N>4 You will get an error
```

Example: ON (N+7)*2 GOSUB 1000,2000

(see GOTO and GOSUB)

IF relational exp THEN statement n

IF relational exp THEN BASIC statement (Direct)

The IF statement is used to control the sequence of program statements to be executed, depending on specific conditons. If the relational expression given in the IF is "true", then control is given to the statement

number declared after the THEN. If the relational expression is "false", program execution continues at the statement following the IF statement.

It is also possible to provide a BASIC statement after the THEN in the IF statement. If this is done, the relational expression is true, the BASIC statement will be executed and the program will continue at the statement following the IF statement.

When evaluating relational expressions, arithmetic operations take precedence in their usual order, and the relational operators are given equal weight and are evaluated last.

Example: 5+6*5>15*2 evaluates to be true

The Relational Operators

=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Examples:

```
110 IF A B+3 THEN 160
180 IF A=B+3 THEN PRINT "VALUE A", A
190 IF A B THEN T1=B
```

Input/Output Statements

Any INPUT or PRINT statement may be followed with a "#N" where "N" is the input or output port number (Ø-7). "N" may be a constant, variable or exp.

The default option (no "#N" specified) is PORT#1, the control port. (Or as specified with the PORT statement) - a comma (,) must follow the port number if anything follows the command.

Example:

```
INPUT #3,A
PRINT #7;"TEST"
PRINT #7
```

INPUT var (var...,var)

The INPUT statement allows users to enter data from the terminal during program execution.

Example:

```
INPUT X - Inputs one numeric value
INPUT X$ - Inputs one string value
```

number declared after the THEN. If the relational expression is "false", program execution continues at the statement following the IF statement.

It is also possible to provide a BASIC statement after the THEN in the IF statement. If this is done, the relational expression is true, the BASIC statement will be executed and the program will continue at the statement following the IF statement.

When evaluating relational expressions, arithmetic operations take precedence in their usual order, and the relational operators are given equal weight and are evaluated last.

Example: 5+3<2+4 evaluates to 0; true

The Relational Operators

=	Equal
<>	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Examples:

```
IF A$="Y" THEN 160
IF A=B+1 THEN PRINT "Value A", A
IF A$ <> B$ THEN C=D
```

Input/Output Statements

Any INPUT or PRINT statement may be followed with a "#N" where "N" is the input or output port number (N=1). "N" may be a constant, variable or exp

The default device (no "#" specified) is PORT#1, the control port. For an unspecified port the PORT statement (a comma (,) must follow the port number if anything follows the command.

Example: INPUT #1,A
PRINT #1,"HI"
PRINT #1

INPUT and INKEY$

The INPUT statement allows users to enter data from the terminal during program execution.

Example: INPUT X - Inputs one numeric value
INPUT X$ - Inputs one string value

INPUT X,Y,Z,B$ - Multiple Inputs may be entered, seperated by ",". If the expected number of values are not entered, another "?" will be generated.

INPUT#2,X - Inputs a value from Port #2.

INPUT "ENTER VALUE,X - Prints the message in quotes, then a "?", and waits for input. It stores the inputed value in X.

When the program comes to an INPUT statement, a question mark is displayed on the terminal device. The user then types in the requested data separated by commas and followed by a carriage return. If insufficient data is given, the system prompts the user with '?'. If no data is entered, or if a non-numeric character is entered, the system prompts "RE-ENTER". however, for string variables a null return will be consideres as valid data. Caution: for input A$,B$,C$ - a null response would create three null variables. Only constants can be given in response to an INPUT statement.

PRINT var (Direct)

PRINT "textstring" (Direct)

PRINT ex (Direct)

The PRINT statement directs BASIC to print the value of expression, literal values, simple variables, or text strings on the user's terminal device. The various forms may be combined in the print list by separating them with commas or semicolons. Commas will give zone spacing of print elements, while semicolons will give a single space between elements. If the list is terminated with a semicolon, the line feed/carriage return will be suppressed.

1. PRINT - Skips a line.
2. PRINT A,B,C - Prints the values of A, B, and C, separated into 16 space zones. Use of a ";" in place of the "," would print A, B, and C separated by one space. (No space is generated if a string variable.) A C/R, LF is generated at the end of the line.
3. PRINT "LITERAL STRING" - Prints the characters contained within the quotes
4. PRINT#7,A,B;"LITERAL" - Prints variables A & B and the word "LITERAL" to PORT #7 (the line printer, by convention).

The TAB Function

The TAB function is used in the PRINT statement to cause data to be printed in exact locations. TAB tells BASIC which position to begin printing the next value in the print list. The argument of TAB may be an expression.

Example: 110 PRINT TAB(2),B,TAB(2*R),C$

INPUT X,Y,Z,$$ - Multiple inputs may be entered, separated by ",". If the expected number of values are not entered, another "?" will be generated.

INPUT#F,X - Inputs a value from file #2.

INPUT "ENTER VALUE";X - Prints the message in quotes, then a "?", and waits for input. It stores the inputed value in X.

When the program comes to an INPUT statement, a question mark is displayed on the terminal device. The user then types in the requested data separated by commas and followed by a carriage return. If insufficient data is given, the system prompts the user with "??". If no data is entered, or if a non-numeric character is entered, the system prompts "RE-ENTER". However, for string variable a null string will be considered as valid data. Conditions for input #3,$,Z$ - a null response would create three null variables. Only constants can be given in response to an INPUT statement.

PRINT use (Direct)

PRINT "separating" (Direct)

PRINT to (Direct)

The print statement directs BASIC to print the value of expression, literal values, simple variables, or text strings to the user's terminal device. The various forms may be combined in the order listed by separating them with commas or semicolons. Commas will give zone spacing of print elements, while semicolons will give a single space between elements. If the list is terminated with a semicolon, the line feed/carriage return will be suppressed.

1. PRINT - Skips a line.
2. PRINT A,B,C - Prints the values of A, B, and C, separated into 14 space zones. Use of ";" in place of the "," would print A, B, and C separated by one space. (No space is generated if a string variable.) A ";", if is generated at the end of the line.
3. PRINT "LITERAL STRING" - Prints the characters contained within the quotes
4. PRINT#1,A,B;"LITERAL" - Prints variables A & B and the word "LITERAL" to #OUT #1 (the line printer, by convention).

The TAB Function

The TAB function is used in the PRINT statement to cause data to be printed in exact locations. TAB tells BASIC which position to begin printing the next value in the print list. The argument of TAB may be an expression.

Example: 110 PRINT TAB(I);B,TAB(I+5);C

Note: The PRINT positions are numbered one to 72.

Functions

RND

RND(X) produces a set of uniformly distributed pseudo-random numbers. If "X" (the seed) is 0, then each time RND(X) is accessed a different number between 0 and 1 will be returned. If "X"<>0, then a specific random number will be returned each time (the same number each time). RND can be called without an argument, in which case it works as if one had used an argument of Ø.

If you require random numbers other than between 0 and 1, then:

"PRINT INT((B-A+1)*RND(0)+A)"

will yield random numbers ranging between A & B.

TAB

TAB(X) will move the print position to the "Xth" position to the right of the left margin. If the print position is already to the right of the position specified in the TAB command, no spaces will be left and printing (if any) will commence.

Note: The first print position (left margin) is position #1.

INT

INT(X) returns the greatest integer less than X.
Example: INT94.354)=4
Now Note this one: INT(-4.354)=-5

ABS

ABS(X) returns the 'Absolute Value" of X.
Example: ABS(3.44)=3.44
ABS(-3.44)=3.44

SGN

SGN(X) returns the 'sign' (+,-,or)) of X.
Example: SGN(4.5)=1
SGN(-4.5)=-1
SGN(0)=0
SGN(-0)=0

POS

Returns the present position of the printhead. (In effect the inverse of TAB),

```
Example:  PRINT TAB(I);X;
          IF POS)=71 THEN PRINT
```

Note: The POINT positions are numbered one to 7.

Functions

RND

RND(X) produces a set of uniformly distributed pseudo-random numbers. If "X" (the seed) is 0, then each time RND(X) is accessed a different number between 0 and 1 will be returned. If "X"<>0, then a specific random number will be returned each time (the same number each time). RND can be called without an argument, in which case it works as if one had used an argument of 0.

If you require random numbers other than between 0 and 1, then:

"POINT INT((B-A+1)*RND(0)+A)"

will yield random numbers ranging between A & B.

TAB

TAB(X) will move the print position to the "Xth" position to the right of the left margin. If the print position is already to the right of the position specified in the TAB command, no spaces will be left and printing (if any) will commence.

Note: The first print position (left margin) is position #1.

INT

INT(X) returns the greatest integer less than X.

Example: INT(34.334)=4
Now Note: this case: INT(-4.334)=-5

ABS

ABS(X) returns the "Absolute Value" of X.

Example: ABS(3.44)=3.44
ABS(-3.44)=3.44

SGN

SGN(X) returns the 'sign' (+,-,or0) of X.

Example: SGN(4.5)=1
SGN(-4.5)=-1
SGN(0)=0
SGN(-0)=0

POS

Returns the present position of the printhead. (It's office the inverse of TAB).

Example: PRINT TAB(I);X;
IF POS>=11 THEN PRINT

LEN

LEN(X$) returns the number of characters contained in X$.

```
Example:  LEN("TESTING")=7
          LEN("TEST ONE")=8
```

Note: The space does count!

Note: LEN(STR$(X)) = The number of print positions required to print the number X.

ASC

ASC (string or string variable) returns the decimal ASCII numeric value of the first ASCII character within the string.

```
Example:  ASC("?")=63
          ASC("A")=65
          ASC("B")=66
          ASC("Z")=90
          ASC("5")=53
          LET B$="5"-->>ASC(B$)=53
```

CHR$

CHR$(X) returns a single character string equivelent to the decimal ASCII numeric value of X. This is the inverse of the ASC function.

```
Example:  CHR$(63)="?"
          CHR$(65)="A"
          CHR$(66)="B"
          CHR$(53)="5"
```

VAL

VAL(X$) returns the numeric constant equivelent to the value in X$. This is the inverse of the STR$ function.

```
Example:  VAL("12.3")=12.3
          VAL("5E4")=5000
          VAL("TWO")=GENERATES AN ERROR.
          VAL("-12.3")=-12.3
```

STR$

STR$(X) returns the string value of a numeric value. This is the inverse of the VAL function.

```
Example:  A=34567
          LET A$=STR$(A)
          A$ NOW EQUALS "34567"
```

LEN

LEN(X$) returns the number of characters contained in X$.

Example: LEN("FIRST")=5
LEN("TEST ONE")=8

Note: The alpha close (")

Note: LEN(STR$(X)) = the number of print positions required to print the number X.

ASC

ASC (string or string variable) returns the decimal ASCII numeric value of the first ASCII character within the string.

Example: ASC("0")=48
ASC("A")=65
ASC("B")=66
ASC("Z")=90
ASC("a")=97
LET B$="a"⟶?ASC(B$)=97

CHR$

CHR$(X) returns a single character string equivalent to the decimal ASCII numeric value of X. This is the inverse of the ASC function.

Example: CHR$(97)="a"
CHR$(65)="A"
CHR$(66)="B"
CHR$(53)="5"

VAL

VAL(X$) returns the numeric constant equivalent to the value in X$. This is the inverse of the STR$ function.

Example: VAL("12.3")=12.3
VAL("3E4")=3000
VAL("TWO")=0 NUMERALS AT FIRST.
VAL("-12.3")=-12.3

STR$

STR$(X) returns the string value of a numeric value. This is the inverse of the VAL function.

Example: A=345.6
LET A$=STR$(A)
A$ NOW EQUALS "345.6"

LEFT$

LEFT$(X$,N) returns a string of characters from the leftmost to the Nth characters in X$.

```
Example:  X$="ONE,TWO,THREE"
          LET A$=LEFT$(X$,6)
          A$ NOW EQUALS "ONE,TW"
```

RIGHT$

RIGHT$(X$,N) returns a string of characters from the Nth position to the left of the rightmost character, through the rightmost character.

```
Example:  X$="ONE,TWO,THREE"
          A$=RIGHT$(X$,9)
          A$ NOW EQUALLS "TWO,THREE"
```

MID$

MID$(X$,X,Y) returns a string of characters from X$ beginning with the Xth character from the left, and continuing for Y characters from that point. Y is optional. If Y is not specified, then the string returned is from the Xth character to the right of the beginning (left side) of the string through the end of the string.

```
Example:  X$="ONE,TWO,THREE"
          A$=MID$(X$,3,10)
          A$ NOW EQUALS "E,TWO,THREE"
```

PEEK

PEEK(X) returns, in decimal, the value contained in (decimal) memory location X.

```
Example:  LET A=PEEK(255)
```

A will now contain the decimal value contained in memory location 255_{10}.

POS

POS returns, in decimal, the current position of the print head or cursor. The first position (left margin) is position #1.

Mathematical Functions

Function	Interpretation
SIN(X)	Returns the SINE of X
COS(X)	Returns the COSINE of X
TAN(X)	Retruns the TANGENT of X
ATAN(X)	Returns the angle, in radians, that is the arc tangent of X
LOG(X)	Returns the NATURAL LOGARITHM of X

EXP(X) Returns the base of NATURAL LOGARITHMS raised to the Xth power (this is the inverse of LOG(X).)

SQR(X) Returns the SQUARE ROOT of X

Note: For these TRANCENDENTAL FUNCTIONS ONLY, the accuracy is stated to seven (7) significant digits, and the accuracy of the seventh digit, or 1E-7 (whichever is greater) is not guaranteed!

Note: For SIN, COS,& TAN the argument is in radians (not degrees).

USER

LET A=USER(X) transfers program control to a USER-written machine language program. Program control branches to the memory location pointed to by memory location $67 and $68. X is a numeric expression which is then stored in a 7 byte series beginning at a memory location pointed to by $5D and $5E (this value may be modified by the USER-written machine language program to act as a 'Data Output' from the program, or as an indicator that "something was done"). The USER program must terminate with a $39, thereby transferring program control back to the BASIC interpreter. Additionally, A is now set equal to the value stored in the 7 byte series stored in a memory location pointed to by $5D and $5E. ($67 denotes hex location ØØ67)

Note that when BASIC is loaded, memory locations $67 and $68 point to a location containing $39, so the USER function will just return control to the BASIC interpreter. You will have to modify memory locations $67 and $68 using the POKE or PATCH command in order to use this function.

Warning: It would be easy to inadvertantly modify the BASIC interpreter, its program, and/or its data using this function. Make sure you understand the machine level implications before using it!! Also, note that if your USER-written machine language program does not end with a $39 (RTS), your function will Bomb, your program will Bomb and BASIC will Bomb --- All is Lost!

DEF FN(X)

DEF FN LETTER (VARIABLE) = EXPRESSION

This function allows the user to create his very own functions. The LETTER is any alphabetic character. This names the function (I.E. you could have, say, three functions named FNA, FNB, and FNC). The VARIABLE is a Non-Subscripted numeric variable. This is essentially a "dummy" variable (or place holder)... This will be apparent shortly. The "Expression" is any valid expression. Note that the "variable" must be enclosed within parenthesis.

For example, study the following sample program:

```
10 DEF FNA(X)=3.14*X
20 DATA 5,6,7,0
30 READ X
```

```
40 IF X=0 THEN END
50 PRINT FNA(X)
60 GOTO 30

RUN

78.5
113.04
153.86

READY
```

As you can see, the dummy variables were replaced with the variables you actually wished to use at the time the function was used.

Note: You may not define the same function greater than once per program, and a function must be defined before it is called.

```
40 IF X=0 THEN END
50 PRINT FNA(1)
60 GOTO 10

RUN

 18.5
 113.04
 153.86

READY
```

As you can see, the dummy variables were replaced with the variables you actually wished to use at the time the FN notion was used.

Note: You may not define the same function greater than once per program, and a function must be defined before it is called.

APPENDIX A

INSTRUCTION SUMMARY

COMMANDS	STATEMENTS		FUNCTIONS	
LIST &	REM	END	ABS	LEN
RUN	DIM*	GOTO*	INT	ASC
NEW	DATA	ON...GOTO*	RND	SQR
SAVE	READ	ON...GOSUB*	SGN	EXP
LOAD	RESTORE	IF...THEN*	USER	LOG
PATCH &	LET*	INPUT	TAB	VAL
APPEND	FOR	PRINT*	PEEK	CHR$
DIGITS &	NEXT	PATCH*	SIN	STR$
LINE &	STOP	RETURN	COS	LEFT$
CONT &	GOSUB*	POKE*	TAN	RIGHT$
PORT &		DEF	FNX	MID$
TRACE ON &			POS	ATAN
TRACE OFF &				

( )* Flags STATEMENTS that may be used in the direct mode (no statement numbers)

( )& Flags COMMAND that may be used in programs

MATH OPERATORS	RELATIONAL OPERATORS	
↑ Exponentiate	=	Equal
-(unary) Negate	<>	Not Equal
* Multiplication	<	Less Than
/ Division	>	Greater Than
+ Addition	<=	Less Than or Equal
- Subtraction	>=	Greater Than or Equal

Line Numbers - May be from 1 thru 9999

Variables - May be single character alphabetic or single character alphabetic followed by one integer 0 thru 9 or $

Backspace - Is a Control 0

Line Cancel - Is a Control X

Panic Button - Typing a Control C should bring Basic back to the READY mode regardless of what the Basic User program is doing.

Lines - Each line may contain multiple statements. Each statement is separated from the other with a : .

APPENDIX B

ERROR MESSAGES

1. Error #_____ in line #_____

   A. If line # = 0000, error was in direct execution statement

2. Error Codes

   1. Oversize variable (over 255) in TAB, CHR, subscript or "ON"
   2. Input error
   3. Illegal character or variable
   4. No ending " in print literal
   5. Dimensioning error
   6. Illegal arithmetic
   7. Line number not found
   8. Divide by zero attempted
   9. Excessive subroutine nesting (max is 8)
   10. RETURN W/O prior GOSUB
   11. Illegal variable
   12. Unrecognizable statement
   13. Parenthesis error
   14. Memory full
   15. Subscript error
   16. Excessive FOR loops active (max is 8)
   17. NEXT "X" w/o FOR Loop defining "X"
   18. Misnested FOR-NEXT
   19. READ statement error
   20. Error in ON statement
   21. Input Overflow (more than 72 characters on Input line)
   22. Syntax error in DEF statement
   23. Syntax error in FN error, or FN called on Function not defined
   24. Error in STRING Usage, or mixing of numeric and string variables
   25. String Buffer Overflow, or String Extract (in MID$,LEFT$, or RIGHT$) too long
   26. I/O operation requested to a port that does not have an I/O card installed
   27. VAL function error - string starts with a non-numeric value.
   28. LOG error - an attempt was made to determine the log of a negative number.

APPENDIX C - SAVING AND LOADING PROGRAMS

SAVE

The SAVE command allows the user to dump the current BASIC program to either cassette or paper tape. Using the SAVE command is similar to the P command of MIKBUG® - punch on/off commands are automatically sent to the recording device. When using a SWTPC AC-30 cassette interface either the MANUAL or AUTOMATIC motor control mode can be used. Turning the recorder to RECORD and typing a SAVE followed by a carriage return will save a copy of the program on tape. The SAVE command dumps the entire BASIC buffer to tape - line numbers such as SAVE 10-20 can not be entered to transfer only a portion of the program to tape. The program in the buffer that is saved is left intact during a save operation.

A single letter file name may be given to a particular program. This name will be punched to tape along with the program. For example, the command SAVE B will save the current program in memory on tape with the file name B. A file name is not necessary.

LOAD

The LOAD command allows for the entering of previously recorded BASIC programs through either cassette or paper tape. The LOAD command is similar to the L command of MIKBUG® - reader on/off commands are automatically sent. Typing LOAD followed by a carriage return will transfer the program from tape to the BASIC buffer. The buffer is automatically cleared at the beginning of a LOAD command.

A single letter file name may also be used with the LOAD command. For example, LOAD B will start the tape reader and load only the program saved with a SAVE B command. Omitting the file name will load whatever program is on the tape to memory as long as it was saved with either a SAVE or SAVE (filename) command. When using the LOAD (filename) command the tape reader should be locked in the READER ON mode.

NOTE: The SAVE and LOAD commands assume that the punch/read device is set up to decode automatic reader/punch/on/off. If your particular unit is not automatic the reader or punch should be turned on manually before the carriage return is entered after the respective LOAD or SAVE command.

APPEND

The APPEND command is identical to the LOAD command except that the current BASIC buffer is not cleared.

The SAVE, LOAD and APPEND commands can all be used to work on any port. If for example your cassette recording device is on the ACIA port three a SAVE #3 command would be used.

APPENDIX D

ASCII Hexadecimal to Decimal Conversion Table

CHARACTER	HEXADECIMAL	DECIMAL	CHARACTER	HEXADECIMAL	DECIMAL	CHARACTER	HEXADECIMAL	DECIMAL
NUL	00	000	+	2B	043	V	56	086
SOH	01	001	,	2C	044	W	57	087
STX	02	002	-	2D	045	X	58	088
ETX	03	003	.	2E	046	Y	59	089
EOT	04	004	/	2F	047	Z	5A	090
END	05	005	0	30	048	[	5B	091
ACK	06	006	1	31	049	\	5C	092
BEL	07	007	2	32	050	]	5D	093
BS	08	008	3	33	051	^	5E	094
HT	09	009	4	34	052	_	5F	095
LF	0A	010	5	35	053	`	60	096
VT	0B	011	6	36	054	a	61	097
FF	0C	012	7	37	055	b	62	098
CR	0D	013	8	38	056	c	63	099
SO	0E	014	9	39	057	d	64	100
SI	0F	015	:	3A	058	e	65	101
DLE	10	016	;	3B	059	f	66	102
DC1	11	017	<	3C	060	g	67	103
DC2	12	018	=	3D	061	h	68	104
DC3	13	019	>	3E	062	i	69	105
DC4	14	020	?	3F	063	j	6A	106
NAK	15	021	@	40	064	k	6B	107
SYN	16	022	A	41	065	l	6C	108
ETB	17	023	B	42	066	m	6D	109
CAN	18	024	C	43	067	n	6E	110
EM	19	025	D	44	068	o	6F	111
SUB	1A	026	E	45	069	p	70	112
ESC	1B	027	F	46	070	q	71	113
FS	1C	028	G	47	071	r	72	114
GS	1D	029	H	48	072	s	73	115
RS	1E	030	I	49	073	t	74	116
US	1F	031	J	4A	074	u	75	117
SP	20	032	K	4B	075	v	76	118
!	21	033	L	4C	076	w	77	119
"	22	034	M	4D	077	x	78	120
#	23	035	N	4E	078	y	79	121
$	24	036	O	4F	079	z	7A	122
%	25	037	P	50	080	{	7B	123
&	26	038	Q	51	081	|	7C	124
'	27	039	R	52	082	}	7D	125
(	28	040	S	53	083	~	7E	126
)	29	041	T	54	084	DEL	7F	127
*	2A	042	U	55	085			

ASCII Hexadecimal to Decimal Conversion Table

CHARACTER	HEXADECIMAL	DECIMAL
NUL	00	000
SOH	01	001
STX	02	002
ETX	03	003
EOT	04	004
ENQ	05	005
ACK	06	006
BEL	07	007
BS	08	008
HT	09	009
LF	0A	010
VT	0B	011
FF	0C	012
CR	0D	013
SO	0E	014
SI	0F	015
DLE	10	016
DC1	11	017
DC2	12	018
DC3	13	019
DC4	14	020
NAK	15	021
SYN	16	022
ETB	17	023
CAN	18	024
EM	19	025
SUB	1A	026
ESC	1B	027
FS	1C	028
GS	1D	029
RS	1E	030
US	1F	031
SP	20	032
!	21	033
"	22	034
#	23	035
$	24	036
%	25	037
&	26	038
'	27	039
(	28	040
)	29	041
*	2A	042
+	2B	043
,	2C	044
-	2D	045
.	2E	046
/	2F	047
0	30	048
1	31	049
2	32	050
3	33	051
4	34	052
5	35	053
6	36	054
7	37	055
8	38	056
9	39	057
:	3A	058
;	3B	059
<	3C	060
=	3D	061
>	3E	062
?	3F	063
@	40	064
A	41	065
B	42	066
C	43	067
D	44	068
E	45	069
F	46	070
G	47	071
H	48	072
I	49	073
J	4A	074
K	4B	075
L	4C	076
M	4D	077
N	4E	078
O	4F	079
P	50	080
Q	51	081
R	52	082
S	53	083
T	54	084
U	55	085
V	56	086
W	57	087
X	58	088
Y	59	089
Z	5A	090
[	5B	091
\	5C	092
]	5D	093
^	5E	094
_	5F	095
`	60	096
a	61	097
b	62	098
c	63	099
d	64	100
e	65	101
f	66	102
g	67	103
h	68	104
i	69	105
j	6A	106
k	6B	107
l	6C	108
m	6D	109
n	6E	110
o	6F	111
p	70	112
q	71	113
r	72	114
s	73	115
t	74	116
u	75	117
v	76	118
w	77	119
x	78	120
y	79	121
z	7A	122
{	7B	123
|	7C	124
}	7D	125
~	7E	126
DEL	7F	127

APPENDIX E

Loading BASIC

This BASIC interpreter is being made available on both paper and cassette tape. Before loading BASIC you must make sure you have at least 8K (ØØØØ thru 1FFF) of RAM memory in your computer system. Load BASIC from either your SWTPC AC-30 Cassette Interface or paper tape reader just as you would any other program. The tapes supplied will load the BASIC interpreter as well as set the program counter addresses AØ48 and AØ49 to ØlØØ, the starting address of the BASIC interpreter. To start type G for "Go to User Program". Should for some reason or another you depress the RESET button on the front panel of the SWTPC 6800 Computer System and wish to re-enter BASIC wihhout losing the program you had earlier stored in memory, reset program counter addresses AØ48 and AØ49 to Ø1Ø3 and type G. Setting the program counter to ØlØØ will get you back into BASIC but you will lose any previously entered programs.

1EE3

APPENDIX F

Memory Map

0000 - 00FF	Input buffer and temporary variable storage
0100 - 1DB0	BASIC interpreter
1EAF - ----	User program

Useful Locations

002A - 002B	Contains the next available memory location after the BASIC program
002E - 002F	Contains the start of the BASIC program
005D - 005E	Contains the address of the current arithmatic value in use during a USER call
0067 - 0068	Contains the pointer for USER
014E - 014F	This location tells BASIC where to start allocating memory for programs and variable storage. Currently this location contains the lowest possible address of 1EAF. If, for example, you desire to store a 100 byte machine code program you can allocate memory from 1EAF to 1FAF by changing 014E to 1FAF.
1EE2	
0150	Contains the number of the port which BASIC will initialize to. This location currently contains 01, the control port.

003E STRING LENGTE

0153 Contains the hex ASCII value of the line delete character. This location is normally a 18 (CTRL. X) but may be changed if desired.

0154 Contains the hex ASCII value that BASIC interprets as a backspace. This location is currently a 0F (CTRL. O). This location can be changed for terminals which generate an automatic cursor left with some other backspace command.

0155 Contains the character echoed to the terminal when a backspace command is entered. This character is currently a 5F, an underline (_) on SWTPC CT-1024 terminals. If desired this character can be changed to a null (00) if your terminal does an automatic cursor left upon sending a backspace command, such as the CT-64 terminal.

0156 Contains the character which BASIC interprets as a break. Currently a 03 (CTRL. C)

Notes: 1.) The last 256 bytes of memory available are used as a string expression buffer and for the machine stack.

2.) A04A - A0FF is used as an index register stack.

Below is a list of the I/O jumps in BASIC for the various ports. For each port the first is the "output character in accumulator A" jump, the second receives input and places it in accumulator A and the third is the initialization routine for a particular type port (ACIA or PIA). This I/O can be changed at the discretion of the user if desired.

```
                        *PORT 0
0106 7E 03CC JMPTAB JMP     OUTACI
0109 7E 03BD        JMP     INACIA
010C 7E 0347        JMP     ACIINZ
                        *PORT 1
010F 7E E1D1        JMP     OUTEEE
0112 7E E1AC        JMP     INEEE
0115 7E 171F        JMP     DUMRTS
                        *PORT 2
0118 7E 03CC        JMP     OUTACI
011B 7E 03BD        JMP     INACIA
011E 7E 0347        JMP     ACIINZ
                        *PORT 3
0121 7E 03CC        JMP     OUTACI
0124 7E 03BD        JMP     INACIA
0127 7E 0347        JMP     ACIINZ
                        *PORT 4
012A 7E 03E2        JMP     OUTPIA
012D 7E 03D7        JMP     INPIA
0130 7E 0352        JMP     PIAINZ
                        *PORT 5
0133 7E 03E2        JMP     OUTPIA
0136 7E 03D7        JMP     INPIA
0139 7E 0352        JMP     PIAINZ
                        *PORT 6
013C 7E 03E2        JMP     OUTPIA
013F 7E 03D7        JMP     INPIA
0142 7E 0352        JMP     PIAINZ
                        *PORT 7
0145 7E 03E2        JMP     OUTPIA
0148 7E 03D7        JMP     INPIA
014B 7E 0352        JMP     PIAINZ
```

APPENDIX G

Notes for Speeding up BASIC

1. Subscripted variables take considerable time; whenever possible use non-subscripted variables.

2. Transcendental functions (SIN,COS,TAN,ATAN,EXP,LOG) are slow because of the number of calculations involved, so use them only when necessary. Also the ↑ operator uses both the LOG and the EXP functions, so use the format A*A to square a number.

3. BASIC searches for functions and subroutines in the source file, so place often called routines at the start of the program.

4. BASIC searches the symbol table for a referenced variable, and variables are inserted into the symbol table as they are referenced, therefore reference a frequently called variable early in the program so that it comes early in the symbol table.

5. Numeric constants are converted each time they are encountered, so if you use a constant often, assign it to a variable and use the variable instead.

APPENDIX H

Notes on Memory Usage in BASIC

1. REM statements use space, so use them sparingly.

2.a. Each non-subscripted numeric variable takes 8 bytes.

b. Each non-subscripted string variable takes 34 bytes.

c. Each numeric array takes 6 bytes plus 6 bytes for each element.

d. Each string array takes 6 bytes plus 32 bytes for each element.

3. An implicitly dimensioned variable creates 10 elements (or 10 by 10). If you do not intend to use all 10 elements, save memory by explicitly dimensioning the variable.

4. Each BASIC line takes 2 bytes for the line number, 2 byte for the the encoded key word, 1 byte for the end of line terminator, 1 byte for the line length, plus one byte for each character following the key word. Memory can be saved by using as few spaces as possible.

5. BASIC reserves the uppermost 256 bytes of memory for stack and buffer use.

SWTPC 8K BASIC has been revised to version 2.0 to improve some of the features of version 1.0 and 1.01. The improvements and changes are as follows:

* An error in line 0 now reports correctly.
* SIN (270 degrees) now gives correct answer.
* VAL will now work with a negative number.
* VAL error is now error #27.
* LOAD will no longer cause problems on loading programs with long lines.
* Getting a random number of 0 will no longer lock the random number generator on 0.
* DIGITS=2 for 0.1 and 0.01 now ok.
* String concatenation over 128 now gives an error message instead of bombing BASIC.
* Certain nested FOR-NEXT loop problems fixed.
* BASIC can now be used on a Motorola Evaluation Module on port 1.
* A NEW command does not reset the port number.
* Interrupts are now allowed.
* String variables are now 32 characters long.
* Rubouts put in PGCNTL and READY messages for improved printout on 300 baud unbuffered terminals.
* All transcendental functions have been speeded up by a factor of two.
* Line lengths are now filed for faster searches.
* The arc tangent (ATAN) function has been added.
* Multiple statement lines are now accepted.
* SAVE and LOAD now work with single letter file labels.
* LINE DELETE, CHAR. DELETE and BREAK characters are now user definable.
* All I/O jumps for different ports are now vectored for ease of change by user.